MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2016/2017

ECE4236 – PARALLEL PROCESSING & PROGRAMMING

(CE, EE, LE, ME & TE)

07 MARCH 2017
9:00 A.M – 11:00 A.M.
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of 7 pages only (including this page).

2. **This is an open-book exam**, so the questions are more in-depth and there are fewer questions than normal exams.

3. There are **THREE (3) QUESTIONS** in this paper. **Answer ALL questions**. All questions carry equal marks (20 marks) and the distribution of the marks for each question is given.

4. All questions should be answered using the C programming language.

5. Write your answers in the Answer Booklet provided.

6. State all assumptions clearly.

Question 1

The following functions are defined:

$$f(x) = 8x^4 + 7x^3 - \sin(x)$$
$$g(y) = \cos(y) + 9y^2 + 5y$$

Figure Q1-1 below illustrates the content of a **"Data.txt"** text file, which contains a set of x and y values. The first line in **"Data.txt"** text file contains the number of x and y elements in the text file. The subsequent lines in **"Data.txt"** text file contain the list of x and y values.

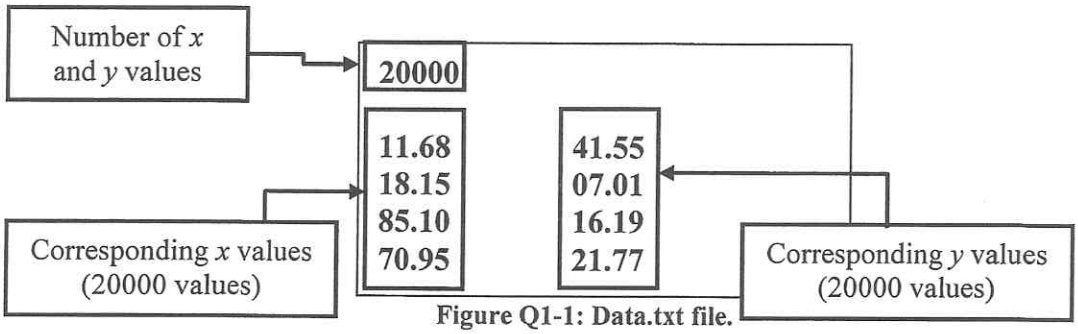

Figure Q1-1: Data.txt file.

(a) Using C programming language with a parallel programming implementation using **POSIX threads**, write a parallel program which computes the numerical values of *f(x)* and *g(y)* for each x and y elements in the **"Data.txt"** text file.

i. In the `main()` function:

- Read the contents of **"Data.txt"** into two x and y global dynamic buffers.
- Create two new *f(x)* and *g(y)* global dynamic buffers based on the number of x and y elements read.
- Launch three (3) threads to compute *f(x)* in parallel. Each thread executes a `FuncFX()` callback function.
- Launch another three (3) threads to compute *g(y)* in parallel. Each thread executes a `FuncGY()` callback function.
- Wait for all six (6) threads to complete before exiting.

Note: *All six (6) threads must run concurrently. There is no need to perform memory clean up at the end of this function.*

[9 marks]

ii. In `FuncFX()` thread callback function:

- Compute the workload distribution based on the thread identifier, number of elements to process and number of assigned threads.
- Compute *f(x)* based on the distributed workload and store the result into the created global dynamic buffer.

[4 marks]

Continued...

iii. In **`FuncGY()`** thread callback function:

- Compute the workload distribution based on the thread identifier, number of elements to process and number of assigned threads.
- Compute *g(y)* based on the distributed workload and store the result into the created global dynamic buffer.

[3 marks]

Note: *Write part (a) as a complete program. Include the appropriate headers.*

(b) Rewrite the program of part (a) using **OpenMP** with **parallel sections** and **nowait** construct. You should be able to identify and apply the correct type of **OpenMP** constructs into your solution.

Note: *There is no need to re-write the dynamic memory creation and data reading from the text file in the* **`main()`** *function. Focus on developing the* ***OpenMP*** *constructs for the iterative processes in the* **`main()`** *function. Include the appropriate header files in your program.*

[4 marks]

Continued...

Question 2

In mathematics, a quadratic equation represents a univariate polynomial equation of the second degree. A general quadratic equation can be described as:

$$ax^2 + bx + c = 0 \quad (2.1)$$

where x represents the unknown variable and a, b and c are the quadratic coefficients ($a \neq 0$). A quadratic equation with real and complex coefficients has two solutions, called roots (x_1 and x_2).

The discriminant, d, is computed as: $d = b^2 - 4ac$. If d is positive, the quadratic equation has two distinct real roots (i.e., $x_1 \neq x_2$) such that:

$$x_1 = \frac{-b+\sqrt{d}}{2a}, x_2 = \frac{-b-\sqrt{d}}{2a} \quad (2.2)$$

If d is zero, the quadratic equation has only one real root (i.e., $x_1 = x_2$) such that:

$$x_1 = x_2 = \frac{-b}{2a} \quad (2.3)$$

If d is negative, the quadratic equation has two distinct complex roots (i.e., $x_1 \neq x_2$) such that:

$$x_1 = \frac{-b}{2a} + \frac{i\sqrt{|d|}}{2a}, x_2 = \frac{-b}{2a} - \frac{i\sqrt{|d|}}{2a} \quad (2.4)$$

Figure Q2-1 illustrates the content of a text file, *quad.txt*, which contains a set of quadratic coefficients. The first row element in *quad.txt* represents the number of coefficients (a, b and c) rows in this file. The second onwards contains the coefficient content. Each row contains a set of a, b, and c coefficients.

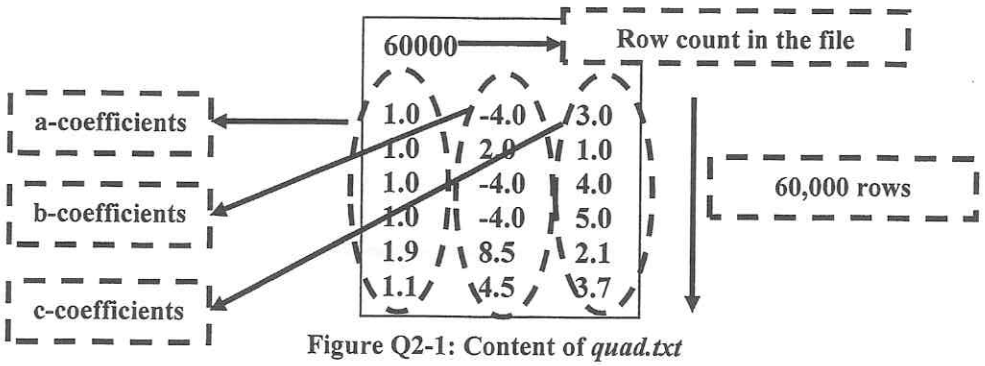

Figure Q2-1: Content of *quad.txt*

Continued...

Using the C programming language with a parallel programming implementation using **Windows (Win32) Threads:**

(a) Write a thread function to compute the quadratic roots of each row coefficient in *quad.txt* file.
   i. Only the first thread is permitted to access the *quad.txt* file.
   ii. This thread reads coefficients of each row from *quad.txt* file and computes the discriminant, d, which is then transmitted to the subsequent thread along with the a and b coefficients.

[6 marks]

(b) Continuing from part (a), the second thread receives the computed d and a and b coefficients (per row) from the first thread and computes the root values.
   i. This thread computes x_1 and x_2 based on the computed value of d.
   ii. Note: If $d < 0$, the roots are calculated as x_{1_real}, x_{1_img} and x_{2_real}, x_{2_img}.
   iii. The computed root values are then transmitted to the third thread.

[8 marks]

(c) Continuing from part (b), the third thread receives the computed root values from the second thread and writes these root values into a new text file, *roots.txt*. Figure Q2-2 illustrates a sample content of the computed root value in the *roots.txt* file.

[6 marks]

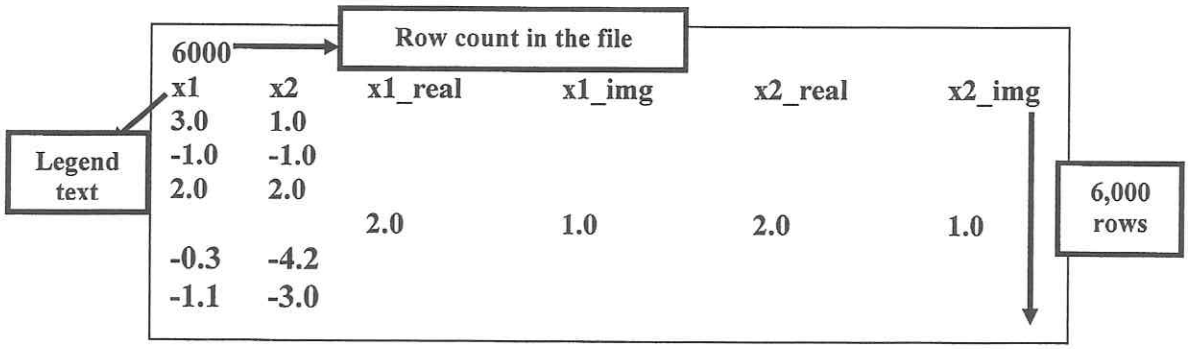

Figure Q2-2: Content of *roots.txt*

Write parts (a), (b) and (c) as a single thread function. Use a **parallel pipeline structure** in reading the coefficients from the file, computing the roots and writing the computed roots into file.

Figure Q2-3 describes the partial code listing of the algorithm. Use the global variables in this figure to assist you in completing the thread function. You can opt to create additional global and/or local variables in the thread functions to complete the algorithm. As the `main()` function is already provided, there is no need to write one.

Continued...

```
#include <windows.h>
#include <stdio.h>
#include <math.h>

#define NUM_THREADS 3

int g_Count = 0;
float a_coeff = 0.0, b_coeff = 0.0, c_coeff = 0.0,
x1 = 0.0, x2 = 0.0, disc = 0.0, disc_2 = 0.0;
float x1r = 0.0, x1i = 0.0, x2r = 0.0, x2i = 0.0;

HANDLE g_CountEvt;
HANDLE g_CountEvtAck[NUM_THREADS - 1];
HANDLE g_ThreadEvt[NUM_THREADS];

// Parts (a), (b) and (c) - To be completed.
DWORD WINAPI threadFunc(LPVOID pArg) {}

int main(){
  int i = 0;
  g_CountEvt = CreateEvent(NULL, TRUE, FALSE, NULL);
  for(i = 0; i < (NUM_THREADS - 1); i++){
     g_CountEvtAck[i] = CreateEvent(NULL, FALSE, FALSE,
                                    NULL);
  }

  for(i = 0; i < NUM_THREADS; i++){
     g_ThreadEvt[i] = CreateEvent(NULL, FALSE, FALSE,
                                  NULL);
  }

  HANDLE hThread[NUM_THREADS];
  int threadIDs[NUM_THREADS];
  for(i = 0; i < NUM_THREADS; i++){
     threadIDs[i] = i;
     hThread[i] = CreateThread(NULL, 0, threadFunc,
                               &threadIDs[i], 0, NULL);
  }
  WaitForMultipleObjects(NUM_THREADS, hThread, TRUE,
                         INFINITE);
  return 0;
}
```

Figure Q2-3: Partial code listing for computing the roots of a set of quadratic equation coefficients.

Continued...

Question 3

An entry-wise NAND operation between two arrays of equal size is denoted as

$$\begin{aligned}\mathbf{A} \uparrow \mathbf{B} &= [a_0 \quad a_1 \quad \dots \quad a_{n-1}] \uparrow [b_0 \quad b_1 \quad \dots \quad b_{n-1}] \\ &= [a_0 \uparrow b_0 \quad a_1 \uparrow b_1 \quad \dots \quad a_{n-1} \uparrow b_{n-1}] \end{aligned} \tag{1.1}$$

Using the C programming language with a **Message Passing Interface (MPI)** parallel programming implementation, write a program to implement the NAND operation between two sets of arrays based on the following requirements:

(a) Only the root rank (or first node) has access to the input content, which are stored as natural numbers in two separate text files. The number of elements in each text file is appended at top of the file. The content of both text files are of equivalent length.

i. The root rank reads the content of each text file into two heap arrays.

ii. The size of each array is distributed to the other nodes.

[7 marks]

(b) Continuing from part (a), the root rank distributes the content of both arrays to the other nodes using the **`MPI Scatter`** and **`Gather`** routines.

i. The division of input array elements among the nodes factors in remaining elements as well.

ii. The root node scatters the content of both input arrays to the other nodes.

iii. Each node (including the root) will then compute and store the NAND between elements of the array.

[10 marks]

(c) Continuing form part (b), the root rank then gathers results of the NAND operation from the other nodes to be saved into a new file.

[3 marks]

Note: *Write a complete program for parts (a) – (c). Include the required header files and perform the necessary clean up at the end of code. Marks will be deducted if you attempt to use alternative techniques to* **`MPI Scatter`** *and* **`Gather`**.

End of Paper